I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

P U B L I C A T I O N
I N T E R N E
N° 1871

DECOUPLED THERMAL SIMULATION

SURENDRA GUNTUR , PIERRE MICHAUD

ISSN 1166-8687

I R I S A

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Decoupled Thermal Simulation

Surendra Guntur , Pierre Michaud

Systèmes communicants
Projets CAPS

Publication interne n ° 1871 — November 2007 — 29 pages

Abstract:

Small transistors and high clock frequency have resulted in high power density, which makes temperature a strong constraint in today's microprocessor design. For maximizing performance, the thermal design power must be set according to average, instead of worst case, conditions. Consequently, current processors feature temperature sensors and throttling mechanisms to keep the chip temperature at a safe level. To study future thermally-constrained processors and systems, researchers and engineers use cycle-accurate performance simulators modeling power consumption and temperature. Cycle-accurate simulators are relatively slow and make it difficult to study long-term thermal behaviors that may require to simulate several minutes or even hours of processor execution. Sampling or phase analysis cannot be applied directly in this case because temperature depends on all past energy events. We propose a partial solution to this problem, which consists in decoupling cycle-accurate simulations and thermal ones. Temperature-unaware cycle-accurate simulation is used to generate an energy trace representing the complete execution of an application. Phase analysis can be used to decrease the trace generation time and make compact traces. Temperature and thermal-throttling are simulated in a separate thermal simulator that reads energy traces. The thermal simulator is faster than the cycle-accurate one and can be used to explore, with the same energy trace, parameters that are not modeled in cycle-accurate simulation.

Key-words: Temperature-constrained processor, performance, energy, simulation, program phase reuse.

(Résumé : tsvp)

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Simulation thermique découplée

Résumé : Les processeurs récents ont une densité de puissance électrique très élevée qui résulte principalement de la miniaturisation des transistors et de l'augmentation de la fréquence d'horloge. De ce fait, la température est une contrainte importante pour la conception des processeurs. Afin d'obtenir un processeur le plus performant possible sous contrainte thermique, on dimensionne la consommation électrique de plus en plus en fonction des conditions moyennes au lieu des conditions extrêmes. Cela nécessite d'intégrer sur la puce des capteurs de temperature et des mécanismes de contrôle permettant de diminuer la dissipation de chaleur lorsque c'est nécessaire pour maintenir la température en deçà de la limite. Afin d'étudier les futurs processeurs et systèmes sous contrainte thermique, les chercheurs et les ingénieurs utilisent des simulateurs microarchitecturaux modélisant la consommation électrique et la température. Ces simulateurs sont relativement lents et ne sont pas appropriés à l'étude de certains comportements nécessitant de simuler plusieurs minutes, voire plusieurs heures d'exécution. L'échantillonnage ou l'analyse de phase de programme ne peuvent pas être utilisés directement, car la température à un instant donné dépend de tous les évènements passés. Nous proposons une solution partielle à ce problème, qui consiste à découpler la simulation microarchitecturale et la simulation thermique. La simulation microarchitecturale est utilisée pour générer une trace d'énergies représentant l'exécution complète d'une application. L'analyse de phase permet de réduire le temps de génération de la trace et d'obtenir une trace compacte. La température et les mécanismes de contrôle sont simulés dans un simulateur thermique séparé. Le simulateur thermique est plus rapide que le simulateur microarchitectural et peut servir à explorer, avec une même trace d'énergies, les paramètres qui ne sont pas modélisés dans le simulateur microarchitectural.

Mots clés : Processeur sous contrainte de température, performance, énergie, simulation, réutilisation de phase de programme.

1 Introduction

Smaller feature sizes and constraints on voltage scaling have resulted in very high power density in current high-performance microprocessors. Consequently, temperature has become a strong constraint and will remain so unless a major technology change occurs. For maximizing performance, the thermal design power will be set more and more according to average, instead of worst case, conditions. In these conditions, it is possible to hit the temperature limit depending on applications characteristics and ambient temperature. Current processors feature thermal sensors to monitor temperature, in order to keep it below the limit. When necessary, the processor power dissipation is throttled, which generally decreases performance.

Researchers are exploring ways to minimize the performance loss due to thermal throttling, for instance with activity migration (or more generally solutions that spread heat over a larger area) [30, 48, 22, 38, 44, 6, 45, 8, 24, 39, 32, 13, 34, 7], with temperature-aware OS scheduling [41, 2, 38, 31, 25, 26, 9], with more efficient throttling mechanisms [23, 3, 47, 11, 48, 49, 50], with temperature-aware floorplanning, [48, 12, 42, 20, 35], with temperature-aware compilation [37, 36, 43] etc. The goal of these techniques is to increase performance under a fixed temperature limit. To study whether and how these techniques should be implemented in future thermally-constrained processors, researchers need simulators.

The conventional approach for temperature-aware cycle-accurate simulation incorporates an energy model (e.g. Wattch [4]) and a thermal model (e.g. ATMI [1] or HotSpot [48]) into a cycle-accurate microarchitecture simulator (e.g. Simplescalar [5]) as shown in figure 1. However, cycle-accurate simulators are generally too slow to simulate more than a few seconds of execution. Yet, thermal behaviors may necessitate to simulate several minutes or even hours of execution in order to be observed. The usual methods to speed-up cycle-accurate simulators, like sampling or phase classification, cannot be used directly in this case because temperature at a given time depends on all past energy events.

We propose decoupled thermal simulation (DTS) as a partial solution to this problem. Temperature-unaware cycle-accurate (TUCA) simulation is used to generate an IPC/energy trace representing the complete execution of an application. Phase analysis can be used to decrease the trace generation time and make compact traces. Temperature and thermal-throttling are simulated in a separate thermal simulator that reads IPC/energy traces. The thermal simulator is faster than the cycle-accurate one and can be used to explore, with the same trace, parameters that are not modeled in cycle-accurate simulation.

This report is organized as follows. Section 2 explains the problem we are trying to solve. Related work is mentioned in Section 3. Section 4 describes decoupled thermal simulation and phase substitution. Section 5 evaluates the accuracy of phase substitution and its impact on trace generation time. Section 6 presents an example of study that can be done with DTS. Finally, Section 7 concludes this study.

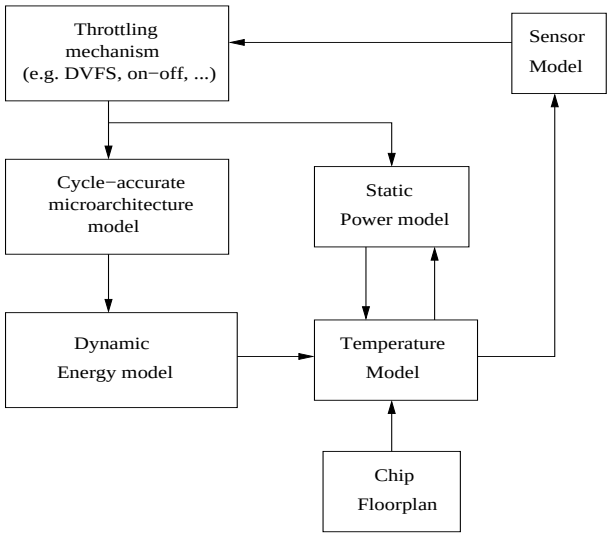

Figure 1: Conventional temperature-aware cycle-accurate simulation.

2 Understanding the problem

Why simulate several minutes of execution to observe thermal behaviors ? The processor heat-sink acts as a reservoir of thermal energy. It is characterized by a certain heat capacity. For example, the specific heat of aluminum is approximately 900 joules per kelvin and per kilogram. An aluminum heat-sink weighting, let's say, $400\,g$ represents a heat capacity $C = 0.4 \times 900 = 360\,J/K$. The time necessary to fill or empty this reservoir depends on the heat-sink thermal resistance R (in kelvins per watt), which itself depends on many parameters (heat-sink dimensions, number of fins, fan speed, etc.). For instance, assuming the heat-sink temperature T is approximately uniform, temperature decreases exponentially with time after power sources have been shut down :

$$\frac{T(t) - T_{amb}}{T(0) - T_{amb}} = e^{-\frac{t}{RC}}$$

where T_{amb} is the temperature of the air hitting the heat-sink. For example, if $R = 0.3\,K/W$, the heat-sink time-constant is $RC = 0.3 \times 360 \approx 108\,s$. It means that an application whose total running time does not exceed a few minutes may not reach a thermal steady state, even if dissipating a constant power. Here, we have assumed that T_{amb} is constant. But the air inside the computer box becomes hotter when the processor dissipates power [51], and the time necessary to reach a true steady state may be much longer than indicated by a quick time-constant calculation.

Can we shorten the simulation by choosing representative execution samples ? The usual solution to speed-up microarchitectural simulations is to use sampling or phase classification [46]. This solution works well for obtaining reasonably accurate estimations of performance and energy consumption. However, it cannot be applied directly to temperature in the general case. Temperature at a given time is a function of all past energy events, mainly events that occurred in a time window corresponding to the time to reach steady state. If we take execution samples without knowing what happened between the samples, we cannot know temperature in general. For example, consider an application with a steady behavior and dissipating a high constant power, but whose running time is too short to reach steady state. If we execute the application after a long period of low activity, the heat-sink is relatively cold, and the temperature when running the application will keep increasing, until thermal throttling triggers and decreases performance more and more as the heat sink gets hotter. In these conditions, defining a representative execution sample seems very difficult, even though the example application has a steady behavior.

3 Related work

The impact of the heat-sink initial temperature was emphasized in [48]. It was proposed to compute the steady-state temperature corresponding to the application average power (i.e., the steady-state temperature we would reach if the application dissipated a constant power) and use this temperature as the initial temperature. It was noted in [50] that a complication arises if one wants to simulate mechanisms that impact the steady state temperature (e.g., thermal throttling methods), which requires iterative simulations. Anyway, these propositions do not apply when an application total running time is too short to reach a steady state, or when the application has distinct execution phases with different power consumptions.

When studying future processors and future platforms that do not exist yet, it is difficult to avoid simulation. However, certain temperature-related studies can be conducted on existing platforms. For instance, temperature can be measured by accessing on-chip thermal sensors [9], by infrared thermal imaging [16, 17], or by correlating power and temperature with processor events via performance counters [2, 28, 10, 21, 25, 19, 51].

4 Decoupled Thermal Simulation

Decoupled thermal simulation (DTS) is based on the observation that power density variations at short time scales have little impact on temperature. If we apply a power density q in a region of the chip where temperature is initially at T_{amb}, temperature T in this region is, for small time values [40]

$$T(t) - T_{amb} = \frac{2q}{k_{si}} \sqrt{\frac{\alpha_{si}}{\pi} t}$$

where k_{si} and α_{si} are the thermal conductivity and thermal diffusivity of silicon respectively. For instance, with $k_{si} = 110\,W/mK$ and $\alpha_{si} = 6\times10^{-5}\,m^2/s$, a power density $q = 2\,W/mm^2$ applied for $t = 10\,\mu s$ generates approximately a temperature increase of 0.5 °C. So in theory, we do not need to know exactly the power density values at each CPU clock cycle to model temperature with reasonable accuracy. If we feed a thermal simulator with a precomputed power density trace, we can work with a timestep much longer than a CPU clock cycle. This way, the ratio $\frac{simulation\ time}{simulated\ time}$ can be much smaller than that of a cycle-accurate microarchitectural simulator, provided we use a fast thermal model like ATMI [33] or HotSpot [48]. Yet, we still need a cycle-accurate simulator to obtain the average power density during a timestep.

DTS is a two-step method :

1. *Trace generation* : An IPC/energy trace representing the complete execution of an application is generated with a temperature-unaware cycle-accurate (TUCA) simulator.
2. *Thermal simulation* : The trace is read by a fast thermal simulator modeling temperature and estimating the performance impact of thermal throttling.

This method saves simulation time when the same trace is reused several times for exploring parameters that are not modeled in the TUCA simulator.

Before going further into the description of DTS, we must mention its limitations. If we change the microarchitecture or any parameter impacting the IPC (instructions retired per cycle) or the dynamic energy consumption in a non straightforward way, we must generate a new trace. Also, not all thermal throttling methods can be simulated with DTS. Decoupling the thermal simulation from the cycle-accurate one is possible only if the throttling method alters performance in a simple way. An *on/off* throttling method that stops the execution for some time can be simulated with DTS, as power density during *off* periods is practically independent from the application characteristics and can be modeled in the thermal simulator, and the impact on performance is straightforward. This is the method we have implemented for this study. Other throttling methods like dynamic voltage/frequency scaling or IPC throttling [14] cannot be easily modeled with DTS. Moreover, the DTS implementation we describe in this report applies to single-thread execution only. We are currently searching ways to apply DTS to multicores, but a complication arises because threads executing concurrently may have complex interactions due to shared resources like caches and memory bandwidth. This is part of our future work. That said, the DTS method we present in this report can be used to do several sorts of studies, in particular activity migration, impact of floorplan on temperature, temperature sensors locations, temperature-aware process scheduling, and package and heat-sink parameters exploration. The rest of this section gives a detailed description of the DTS method we have implemented.

4.1 Trace generation

The TUCA simulator computes the dynamic energy consumed by each resource every cycle. A resource that is idle in the current cycle is assumed to be gated and dissipates a certain

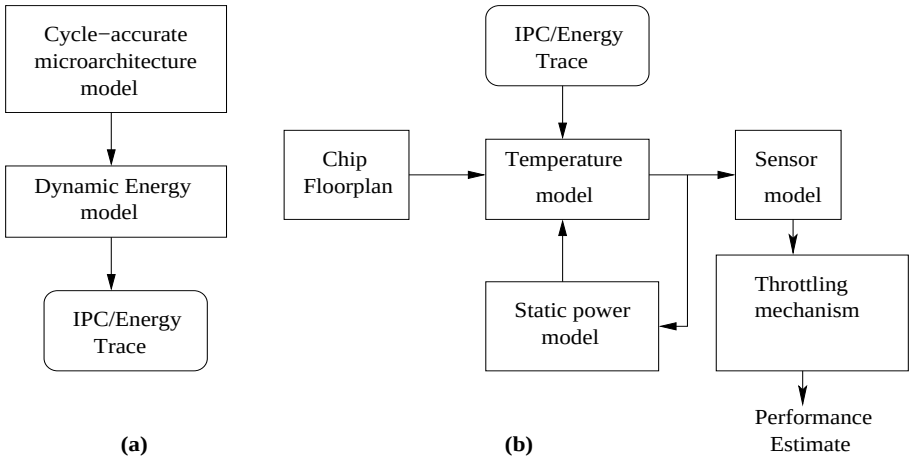

Figure 2: A high level schematic of DTS showing (a) Trace generation and (b) Thermal simulation.

fraction of the dynamic energy that it consumes when it is active. This is called the *gating factor*.

Periodically, at intervals of t_s cycles, the accumulated energy and the interval IPC is stored in the trace file. Figure 3(a) shows the contents of a typical trace. Each row i consists of an IPC value IPC_i and r energy values E_{ij} $(1 \leq j \leq r)$, where r is the number of floorplan blocks (e.g., microarchitectural units). The values represent the IPC of the program and the dynamic energy consumed by each resource over an interval of t_s cycles. Assuming that each value is represented by a floating point number, an uncompressed IPC/energy trace file consisting of N rows (which corresponds to N * t_s simulation cycles) has a size of N * [$(r + 1)$ * sizeof(float)] bytes.

Other variants for generating the trace may also be used. For example, instead of writing into the trace every t_s cycles, it is possible to dump IPC and energy values every fixed number of instructions. Additionally, if the power model assumes that the energy consumed by a resource doing useful work is constant (i.e., it is a statistical model ignoring bits values, like Wattch [4]) then it is sufficient to store the number of cycles N_{ij} during which a resource is doing useful work (cf. Figure 3(b)). Actually, in this case, the power model (i.e., energy EC_j per resource utilization, and gating factors) can be integrated directly in the thermal simulator.

4.2 Thermal simulation

The thermal simulation phase is composed of three main components : thermal model, temperature dependent leakage power estimation and thermal throttling mechanism. Figure

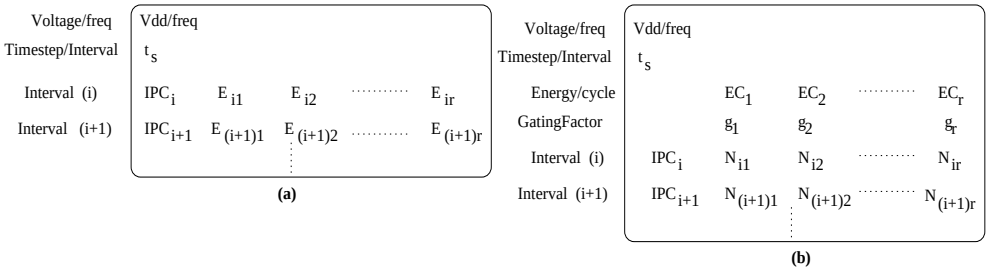

Figure 3: Partial trace contents. (a) Accurate format which consists in storing the energy consumed by each unit over a period of t_s cycles. (b) A simpler more compact format in which resource utilization counts are stored along with the baseline per-resource energy values.

2(b) and algorithm 1 highlight the interaction between each of these components when thermal throttling is used.

Leakage power. Static power dissipation due to leakage currents is a significant fraction of the total power dissipation. In contrast to dynamic power which is application dependent, static power is present even when a circuit is not switching, unless some power gating is implemented. Besides, leakage power, unlike dynamic power, is strongly dependent on temperature [29]. Therefore the leakage power computation is carried out in the thermal simulator. In this report, the temperature dependent variation of leakage power is computed using the model of Liao et. al. [29].

Temperature model. We used the ATMI thermal model for our simulations [1][33]. ATMI models the processor floorplan as a set of rectangles with uniform power density. If the power model gives power numbers at the granularity of main microarchitectural units, as is the case in this study, then each unit is modeled with a single rectangle. If the power model is more detailed, or if we want a more detailed temperature map, we may use several rectangles per unit. In ATMI, the initial thermal state is defined by applying a constant power density in each rectangle for a time sufficiently long to reach a steady state.

Thermal throttling. Modern processors employ thermal throttling mechanisms that reduce power consumption when the temperature exceeds a threshold T_{lim}. This can be done by stopping the clock for a fixed duration t_{off} whenever the hottest on-chip thermal sensor indicates that the thermal limit T_{lim} is exceeded [15]. For instance, in the Intel Pentium 4, t_{off} is a few microseconds [15]. When off, the circuits are in an inactive state and the processor consumes only a fraction the power it consumes when doing useful work. Since the decision to turn-off the processor clock is based on the temperature at the hottest thermal sensor, it is important that the sensors be placed at appropriate locations so as to capture

```
Area(r) = Read_Floorplan(); /* get area of each unit r */
T(x_s, y_s, 0) = T_init; /* start with initial temp */
Initialize_Thermal_Model();
foreach Resource r do
  /* static power at initial temp in each unit */
  StaticEnergy(r, T(x_s, y_s, t), V_dd) = Determine_static_power(r, T(x_s, y_s, t), V_dd);
end
Proc_state = ON
i = 1; Interval_count = 0; AvgIPC = 0;

 while i < Num. trace entries do

  /* Simulate throttling */
  if Proc_state == ON then
    {IPC(i) , DynEnergy(r, V_dd} = Read_Trace_Entry(i);

    /* Active mode */
    TotalEnergy(i, r) = DynEnergy(r, V_dd) + t_s * StaticEnergy(r, T(x_s, y_s, t), V_dd);
    AvgIPC + = IPC(i);
    Interval_count++;
  end
  else
    /* ∃(x_s, y_s) : T(x_s, y_s, t) > T_lim ⇒ processor is Off */
    /* Inactive mode */
    TotalEnergy(i, r) = t_s * InactiveEnergy(r, T(x_s, y_s, t));
    Interval_count++; /* another timestep elapsed*/
    −−DTM_interval;
    if DTM_interval == 0 then
      Proc_state == ON;
    end
  end
  Powerden(r) = Convert_Energy_to_Powerdensity (TotalEnergy(i, r), Area(r), f);
  T(x_s, y_s, t) = Compute_Temperature(Powerden(r), (x_s, y_s), ... );
  i++;

  StaticEnergy(r, T(x_s, y_s, t), V_dd) = Update_Leakage_Power (r, T(x_s, y_s, t));
  if ∃(x_s, y_s) : T(x_s, y_s, t) > T_lim && Proc_state == ON then
    Proc_state = OFF; /* trigger DTM */
    DTM_interval = t_off; /* switch-off for t_off cycles */
  end
end
Overall_IPC = AvgIPC / Interval_count;
```

Algorithm 1: Pseudo-code for the thermal simulator. Timestep is t_s cycles.

the temperature at the truly hot regions of the chip. For this study, we have assumed there is a temperature sensor in each unit.

On-off throttling reduces the power density at all the units and leads to temperature oscillations [34]. The amplitude of the oscillations depends on how long the processor is switched off. A large t_{off} value gives the chip more time to cool resulting in higher amplitudes of oscillation. On the other hand, a small t_{off} leads to higher frequency of oscillations and more frequent triggering of thermal throttling. Figure 4 shows that the amplitude of the oscillations is less than 1^0C and 2.7^0C for t_{off} of $0.1ms$ and $1ms$ respectively. In general, a smaller off period results in lower processor performance degradation [34].

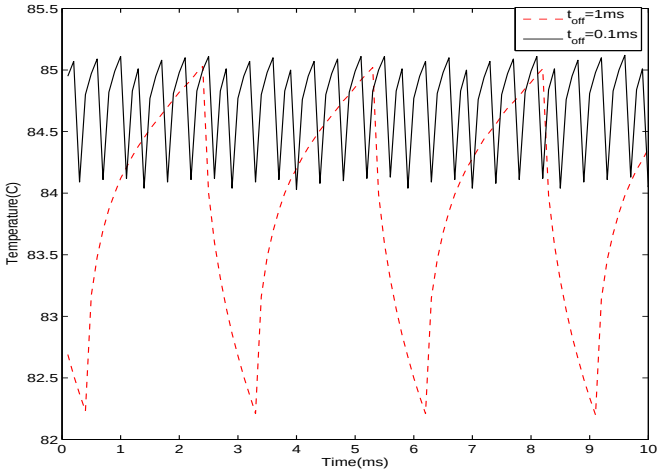

Figure 4: Amplitude of temperature oscillations for two different values of t_{off}.

The thermal simulation phase accepts the IPC/energy trace as input and estimates the temperature profile for the concerned application. Each row i of the IPC/energy trace file represents the IPC and dynamic energy (*DynEnergy*) consumed by every resource r over a period of t_s cycles. Thus, the total energy consumed at each resource over t_s cycles is *total energy* = *dynamic energy* $+t_s\times$ *static power* (see Algorithm 1). The ATMI timestep is set equal to t_s. At each timestep, we compute the power density from the dynamic energy information stored in the trace and from the static power model, and ATMI updates temperature accordingly.

The resulting temperature estimate is used to update the leakage power at each unit for the next timestep. The IPC of the processor during this interval is simply the value read from the trace. Thermal throttling is triggered when the temperature of the hottest thermal sensor exceeds T_{lim}. During this period, the circuits are in an inactive mode and the contents of the trace file are not read (i.e. IPC = 0 and *DynamicEnergy* = 0). The

processor remains turned-off for t_{off} cycles. In general, we take t_{off} as an integral multiple of the timestep t_s, which makes thermal simulation easier.

4.3 Faster and compact trace generation using program phases

Trace generation involves cycle-accurate simulation of the *entire* benchmark and is therefore the most time consuming part of DTS. An alternative approach for trace generation is to sacrifice some accuracy for speed by simulating only representative portions of the program [27, 46]. As already mentioned in Section 2, phase classification cannot be applied directly to temperature because temperature depends on all past energy events. However, it can be used to speed up trace generation and to make more compact traces.

We use the SimPoint tool [18] to identify representative phases and the sequence of phases within a program. The dynamic instruction stream of a program is divided into fixed or variable length intervals I_i and the basic block vector frequency counts (BBV) are obtained via functional simulation for each interval. Unless otherwise mentioned, we use fixed length intervals I_i of 100 million instructions. We find that using variable length intervals does not improve the accuracy of the temperature estimate by much. SimPoint uses the BBV counts and K-means clustering to identify the *simulation points* of the program. It also assigns a phase label P_i to each interval I_i. For better understanding, we shall describe this approach using the example of figure 5 which shows an application consisting of five intervals I_1, ... , I_5. SimPoint identifies three phases P_1, P_2 and P_3 for this application. Interval I_1 is assigned a phase label P_1 while I_4 has a label P_3.

Baseline DTS: The baseline DTS trace generation ignores phases. It simulates every interval with a TUCA simulator, as described in section 4.1. For the example of Figure 5, the overall trace is a concatenation of the individual interval traces in program order i.e. $\{T_1R|T_2R|T_3R|T_4R|T_5R\}$.

Direct Phase Substitution: The idea behind phase substitution is that since intervals having the same phase label (i.e. I_1, I_3 and I_5) are approximately similar in their behavior, only one representative interval (or simulation point) may be simulated in detail (e.g. I_3), and the IPC/energy values of this representative interval may be used for the remaining intervals having the same phase label (i.e. for I_1 and I_5). Once the representative phases have been identified, the simulator processes instructions of an interval I_i either using functional simulation or using detailed cycle-accurate simulation. With regard to figure 5, instructions belonging to intervals I_1 and I_5 are skipped using fast-forwarding, while those belonging to I_2, I_3 and I_4 are executed in the detailed cycle-accurate simulation mode. Thus, the IPC/energy trace file is generated much faster than the baseline policy and consists of values from only these intervals i.e. Trace = $\{T_2R|T_3R|T_4R\}$. For the skipped intervals, only some pointer information is stored (see figure 5(c)). These pointers point to the start and end of the respective simulation points and are used only during thermal simulation. Thus, a program with P phases has IPC/energy entries corresponding to P intervals.

During thermal simulation, the IPC/energy values of I_3 (i.e. T_3R), which is representative of phase P_1 is used for the (skipped) intervals I_1 and I_5 (the pointers in the trace file enable this to be done). This substitution of the IPC/energy values is required since tem-

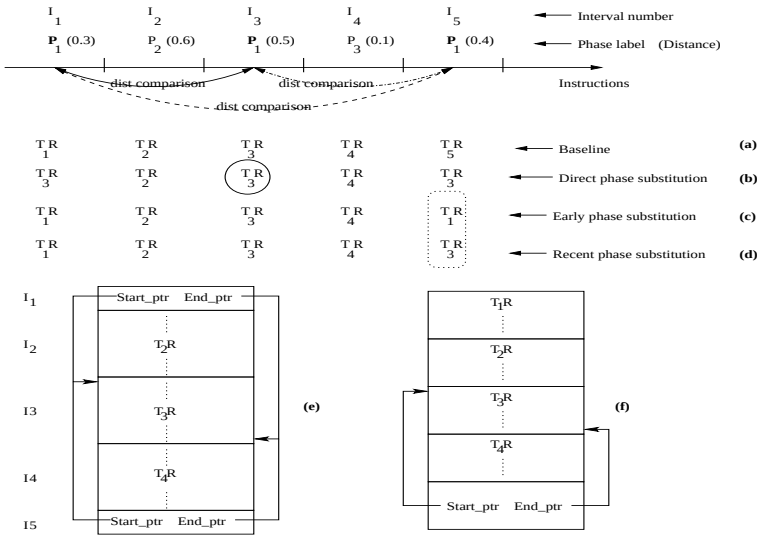

Figure 5: DTS using program phase information with various substitution schemes. The simulation points for phases P_1, P_2 and P_3 are intervals I_3, I_2 and I_4 respectively. Each interval trace T_iR contains IPC/energy values in the format shown in figure 3. Figures (a)-(d) show the "unrolled" values of the trace files used during thermal simulation. Figures (e) and (f) show the format of the trace generated with direct and recent phase substitution respectively. The start and end pointers point to the beginning and end of the interval whose values need to be used during thermal simulation.

perature is a function of all previous energy events. Thus, unlike performance, which can be estimated once the CPI of each phase is available [18], temperature simulation requires that the IPC/energy of *all* intervals be processed in program order. Thus, for the example considered, the IPC/energy values used by the thermal simulator to estimate the temperature behavior of the complete application is essentially the trace $\{T_3R|T_2R|T_3R|T_4R|T_3R\}$.

Selective Phase Substitution: To improve the accuracy of direct phase substitution, specially for benchmarks that have complex phase behavior, one can use the following methods - (i) increase the number of representative simulation points, for example, by changing the parameters of SimPoint and (ii) by being more selective during substitution by simulating more intervals in the detailed mode. In this study, we evaluate the latter "selective phase substitution" approach that takes into account not just the phase labels, but also the cluster distance values associated with each interval. The cluster distance is a measure of how close or similar an interval I_i is with respect to the representative interval (or simulation point) at the cluster center. The smaller the distance value, the greater the similarity. If the phase labels of two intervals I_i and I_j (I_j occurs after I_i in program order) match, and the cluster distances between them is less than or equal to a threshold *Dist_Thresh*, then,

I_j is substituted with the IPC/energy values of I_i (provided trace T_iR is available). Thus, instructions of I_j are skipped using the functional mode of execution. This scheme introduces selectivity by ensuring that substitution is done only when two intervals are relatively similar. Two variants of this scheme are described below. For illustration purposes, we assume that *Dist_Thresh* = 0.1 and describe the method with regard to figure 5.

The first time an interval with a new phase label is encountered, its instructions are executed using cycle accurate simulation. Thus, instructions belonging to intervals I_1 and I_2 are executed in detailed mode resulting in portions of the trace file T_1R and T_2R. Interval I_3 has the same phase label as I_1. However, the cluster distance metric between I_3 (0.5) and I_1 (0.3) is greater than *Dist_Thresh* (0.1). Thus, substitution is not done and instructions of I_3 are executed in the cycle accurate mode. Interval I_4 has a new phase label P_3 which was not encountered before and is executed in detailed mode. Interval I_5 has the same phase label as a previously encountered interval (e.g. I_1 and I_3).
In the **early phase substitution** policy, the IPC/energy values of the earliest encountered phase (that was executed in detailed mode) having the same phase label are used for substitution. Thus, instructions of I_5 are skipped (via functional simulation) and it is assumed to have an IPC/energy trace of I_1 i.e. T_1R.
In the **recent phase substitution** policy, the IPC/energy values of the most recent interval (having same phase label) that was executed in detail is used for substitution. Thus, I_5 uses the trace contents of I_3 (i.e. T_3R) since the distances between these is equal to the threshold.

To prevent values of one particular portion of the program from being propagated all over, we resort to detailed simulation after a a certain number of substitutions. The idea is to use the IPC/energy values of the current program slice for future substitutions. For example, with the early phase substitution policy, the IPC/energy values of the initialization portion of the program I_1 are propagated to all subsequent intervals that have the same phase label and satisfy the distance criterion. This may not be representative of the original program behavior. To prevent this from happening, we allow a maximum of 60 substitutions at a time after which the association between the phase labels and the intervals is cleared. The start reference is changed to the current interval that is yet to be simulated and the methodology is repeated assuming that this is the first interval in the program. Thus, with this approach, IPC/energy values from different portions of the program are used for phase substitution.

5 Phase substitution : accuracy and trace generation time

5.1 Simulation environment

For TUCA simulation, we use the PTScalar [29] simulation infrastructure. This infrastructure is based on the Simplescalar performance simulator [5] and incorporates a dynamic and static power model.

Parameter	**Configuration**
Fetch	4 instr/cycle. 8 entry IFQ. 1 taken branch/cycle.
Branch	Combined, 4K-entry bimodal, 2-level 1K table, 10 bit history, 4K chooser 8 entry RAS. 512 set 4 way BTB
Dispatch	4 instr/cycle, RUU = 64, LSQ = 32
Issue	4 instr/cycle, OoO issue, in-order commit
L1 I/D cache	64KB, 32-byte block, 4 way assoc.
L2 cache	Unified, 4MB, 128B block, 8 way assoc.
Miss latency	1 cycle for L1 I/D-cache, 12 cycle for L2
FUs	3 IALU, 1 IMULT, 1 FPALU, 1 FPMULT,
FU latency/ throughput	IMUL 3/1, IDIV 20/19, Other integer 1/1 FPMULT 4/1, FPDIV 12/12, FPALU 2/1

Table 1: Baseline processor configuration.

The simulator runs on a host machine with an Intel Xeon processor having a 2.8 GHz clock. The baseline configuration that we simulate is a 4-way dynamically scheduled superscalar processor with a floorplan closely resembling the Alpha 21264 processor ([29]). We choose 100-nm technology in our experiments and integrate the ATMI thermal model [1] in this simulator to estimate the temperature behavior. The baseline processor configuration and the ATMI parameters used are shown in Tables 1 and 2. In all our simulations, we used a timestep $t_s = 5 \times 10^5$ cycles, which, with a 5 GHz clock, corresponds to $100\,\mu s$. Parameter t_{off} represents the duration (in cycles) for which the processor is turned-off when thermal throttling is active. Unless stated otherwise, we assumed $T_{lim} = 85$ °C and $t_{off} = 1\,ms$.

Benchmarks. We present results only for a subset of SPEC CPU2000 benchmarks. The benchmarks chosen have widely varying thermal stress properties ranging from medium to high and are simulated to completion. Table 3 shows some statistics including the baseline IPC, *dynamic* energy per instruction (EPI) and time taken to generate the IPC/energy trace. The IPC/energy trace generated with baseline DTS is used as the reference for all comparisons.

It is important to distinguish the *simulated time* from the *simulation time*. The simulation time is the time required to simulate a benchmark (denoted by **Time** in table 3). It is a function of the simulator complexity. The simulated time represents the time it would take to execute the benchmark on the processor we have modeled in the simulator if this processor were real.

Name and units	**Value**
Silicon thermal conductivity; k_1 *(W/mK)*	124.2
Copper thermal conductivity; k_2 *(W/mK)*	400
Silicon thermal diffusivity; $\alpha 1$ *(m^2/s)*	7.1 x 10^{-5}
Copper thermal diffusivity; $\alpha 2$ *(m^2/s)*	1.1 x 10^{-4}
Interface material thermal conductance; h_1 *(W/m^2K)*	8 x 10^4
Heat sink thermal conductance; h_2 *(W/m^2K)*	680.2
Die thickness; z_1 *(mm)*	0.5
Heat sink base plate thickness; z_2 - z_1 *(mm)*	5
Heat sink base plate width; L *(cm)*	7
Supply voltage/frequency; V_{dd} */ Frequency*	1.1V / 5GHz
Ambient temperature; T_{amb}	45^0C

Table 2: Baseline ATMI parameters.

Name	**Input**	**Instr**	**Phase**	**IPC**	**Time**	**EPI**	**Power**	**Sim**
art	110	41.7	9	1.76	63	4.0	35.5	4.7
bzip2	graphic	143.5	12	1.56	266	3.5	27.7	18.3
eon	kajiya	101.2	10	1.83	152	3.7	33.5	11.0
gcc	166	46.9	12	1.50	76	3.9	29.7	6.2
gzip	graphic	103.7	10	1.85	166	3.4	32.0	11.1
lucas	ref	142.3	13	0.50	344	6.5	16.4	56.2
mcf	ref	61.8	12	0.28	347	11.8	16.5	43.8
perl	diffmail	39.9	7	1.52	85	4.1	30.8	5.2

Table 3: Benchmark statistics for the baseline processor configuration including Inputs, number of instructions Instr (in billions) and number of phases in the program, IPC, approximate simulation time to generate IPC/energy trace (in hours), average EPI (nJ/instr), average power (Watts) and simulated time (Sim) in seconds.

Name	**Scheme**	**0.02**	**0.04**	**0.06**	**0.08**	**0.1**	∞
art	Early	104	97	70	70	70	36
	Recent	49	40	37	37	37	36
bzip2	Early	1087	862	782	636	469	177
	Recent	1083	897	739	627	527	177
eon	Early	102	99	99	99	99	99
	Recent	100	99	99	99	99	99
gcc	Early	405	381	332	309	299	44
	Recent	220	166	148	123	118	44
gzip	Early	212	85	77	77	77	69
	Recent	192	95	71	70	70	69
lucas	Early	1307	1275	1080	1048	919	260
	Recent	1118	1021	934	860	805	260
mcf	Early	562	401	206	198	100	65
	Recent	214	153	130	112	100	65
perl	Early	156	31	20	19	18	16
	Recent	28	23	21	21	20	16

Table 4: Number of intervals I_i simulated in detail with Early and Recent phase substitution. The column denoted ∞ represents the case with infinite *Dist_Thresh*.

5.2 Baseline DTS vs. phase substitution based approximation

We first examine the number of program intervals that are subjected to detailed cycle-accurate simulation as a function of *Dist_Thresh* (see Table 4). As the *Dist_Thresh* value increases, the potential for phase substitution improves and the number of intervals I_i subjected to detailed cycle-accurate simulation (during trace generation) reduces. The simulation time for trace generation can be improved further by using checkpoints. This is possible since the intervals to be substituted are known in advance (can be computed offline) once the phase labels and cluster distance values of all the program intervals are available.

When thermal throttling is triggered, the temperature profile of an application changes significantly making it impossible to compare temperature curves with the naked eye. For example, figure 6 shows a snapshot of the temperature curves obtained with baseline DTS and those obtained by the early and recent phase substitution policies. The obvious question that arises is "Which temperature curve best represents the exact behavior?". By looking at the figure, it is hard to conclude if the early or the recent phase substitution policy is better. At some portions, the early policy seems to do better, while at others, the temperature response with recent phase substitution seems more accurate. Thus, appropriate metrics are required for comparing temperature curves in the presence of thermal throttling. Simply computing the temperature difference at various instances in time is not a viable alternative

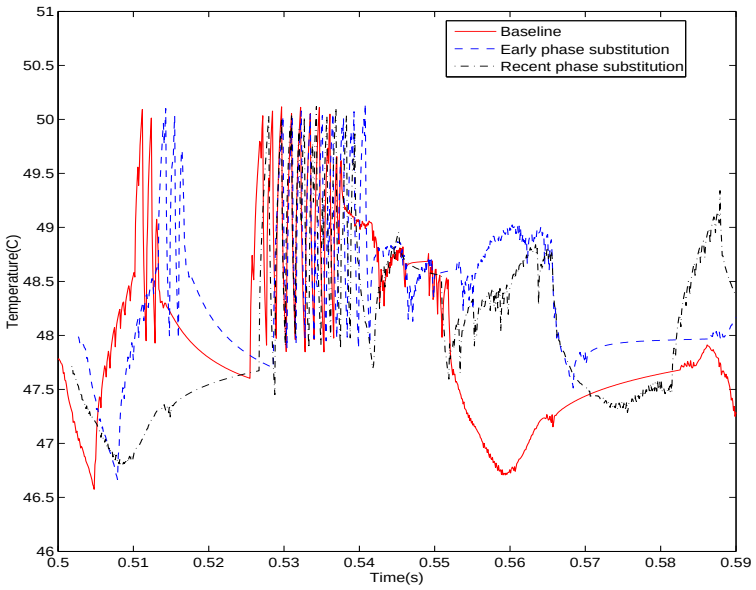

Figure 6: A snapshot of the temperature behavior of bzip2 as estimated by baseline DTS and early/recent phase substitution when thermal throttling is triggered for a hypothetical value of 50°C.

since the difference can be large when the temperature responses, of even identical curves, are shifted.

The metric we choose for this study is the dynamic energy. Since the total energy consumed by a program is a constant, the difference between the energy obtained by DTS and that obtained with approximate simulation (e.g. via direct/early/recent phase substitution) is a reasonable way of quantifying the accuracy of a simulation scheme; the smaller the difference, the greater is the accuracy. This computation can be done once the dynamic energy trace is available.

Table 5 shows the dynamic energy, the approximate trace generation time, and the simulated time of baseline DTS vs. direct phase substitution, without thermal throttling. As expected, SimPoint allows to obtain reasonably accurate performance and energy values while decreasing considerably the simulation time (not counting the time to generate frequency vectors and classify phases). These results are consistent with published results on SimPoint [18]. Table 6 shows the dynamic energy values for different *Dist_thresh* values with early and recent phase substitution. For easier comparison, the energy for baseline DTS and direct phase substitution is also shown. The results indicate that, recent phase substitution is more accurate than early or direct phase substitution. This can be primarily attributed to the fact that a larger fraction intervals are simulated in detail with this scheme. The recent

Name	Baseline DTS			Direct phase substitution		
	Energy	**Time**	**Sim**	**Energy**	**Time**	**Sim**
art	167.8	63	4.7	167.7	2.2	4.71
bzip2	508.7	266	18.3	506.7	7.5	18.75
eon	369.1	152	11.0	369.4	5.1	11.02
gcc	184.7	76	6.2	181.3	2.4	6.0
gzip	356.9	166	11.1	357.3	5.4	11.13
lucas	924.6	344	56.2	920.0	7.0	55.95
mcf	726.4	347	43.8	714.9	3.2	43.07
perl	161.2	85	5.2	162.3	1.9	5.27

Table 5: Total dynamic energy (in Joules), approximate trace generation time (**Time** in hours) and simulated time (Sim in seconds) of the benchmarks for baseline DTS and direct phase substitution (no thermal throttling).

substitution policy for *Dist_Thresh* = 0.1 (which results in the best energy estimate with minimal number of intervals being subjected to detailed simulation) is henceforth referred to as the best-recent phase substitution policy.

However, for temperature, the picture is different. Since phase substitution involves using the IPC/energy values of another interval (instead of the actual values which can only be obtained by detailed simulation), it may introduce significant errors in power density and temperature. In particular, since the program behavior across the concerned intervals is rarely the same (even though they have the same phase label), substitution introduces error in two ways - (i) by approximating the IPC/energy values of interval I_j with those from I_i and (ii) by approximating the number of values substituted. The impact of the former is reflected as a difference in the magnitude of the temperature behavior, while that of the latter as either an elongation or compression of the time axis in the temperature profile. As an example, Figure 7 shows the temperature curve obtained with baseline DTS for the *gcc* benchmark, vs. the temperature curve obtained with direct phase substitution. As can be seen in this example, for benchmarks like *gcc* with complex phase behavior, phase substitution may incur large errors on the simulated temperature, even though overall performance and energy are estimated reasonably accurately. For this kind of application, a finer phase granularity may be necessary. This is confirmed by the results for the best-recent substitution policy in which a larger number of intervals are subjected to detailed simulation.

Table 7 shows the impact of phase substitution on performance in the presence of thermal throttling. We assume on/off throttling with $t_{off} = 200\,\mu s$, and we vary the value of T_{lim} from 70°C to 85°C. Performance is given by the IPC, obtained by dividing the total number of retired instructions by the total simulated time (counting *off* periods). For these simulations, the ambient temperature is $T_{amb} = 45$°C and the heat-sink thermal resistance is $R = 0.8\,K/W$. The initial thermal state is defined with the steady state temperature

Name	**Base**	**Direct**	**Scheme**	**0.02**	**0.04**	**0.06**	**0.08**	**0.1**
art	167.8	167.7	Early	167.7	167.7	167.9	167.9	167.9
			Recent	167.8	167.7	167.8	167.8	167.8
bzip2	508.7	506.7	Early	509.8	510.0	507.2	507.8	508.4
			Recent	508.3	508.5	508.8	511.0	508.1
eon	369.1	369.4	Early	368.9	368.9	368.9	368.9	368.9
			Recent	368.9	368.9	368.9	368.9	368.9
gcc	184.7	181.3	Early	184.2	184.5	184.4	185.2	185.4
			Recent	184.6	184.9	184.2	184.6	184.4
gzip	356.9	357.3	Early	356.5	357.4	356.6	356.5	356.5
			Recent	356.4	357.1	356.9	356.5	356.5
lucas	924.6	920.0	Early	925.1	924.3	922.8	925.6	923.7
			Recent	922.0	924.1	925.3	926.2	926.3
mcf	726.4	714.9	Early	727.6	724.3	718.5	717.5	751.4
			Recent	727.0	728.0	722.4	719.4	724.2
perl	161.2	162.3	Early	160.9	162.0	162.2	161.9	162.0
			Recent	161.1	162.9	161.9	161.9	162.3
Avg	**424.9**	**422.4**	**Early**	**425.0**	**424.8**	**423.5**	**423.9**	**428.0**
			Recent	**424.5**	**425.2**	**424.5**	**424.5**	**424.8**

Table 6: Comparison of the dynamic energy (in Joules) of baseline DTS (Base), direct phase substitution (Direct) and Early/Recent phase substitution. In case of early/recent substitution, the energy for *Dist_thresh* values of 0.02 through 0.1 is shown.

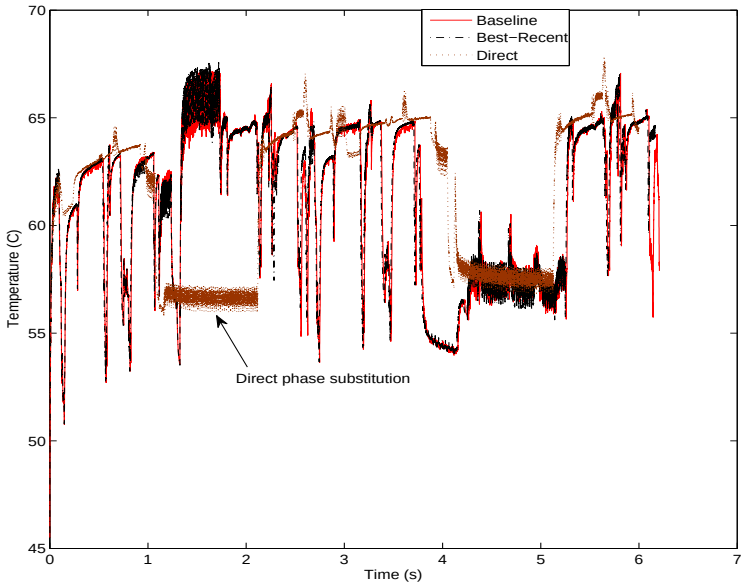

Figure 7: Temperature at the center of the instruction cache for *gcc* with baseline DTS compared to direct phase substitution and best-recent phase substitution Thermal throttling is disabled. The number of intervals simulated in detail are 12 and 118 with direct and best-recent phase substitution respectively.

Name	**Scheme**	**IPC**				
		70°C	**75°C**	**80°C**	**85°C**	∞
art	Base	0.25	0.79	1.27	1.74	1.76
	Best	0.25	0.79	1.27	1.74	1.76
	PS	0.25	0.80	1.27	1.75	1.76
bzip2	Base	0.67	0.97	1.10	1.42	1.56
	Best	0.66	0.98	1.10	1.42	1.56
	PS	0.71	1.02	1.19	1.48	1.52
	PS 92	0.66	0.99	1.10	1.39	1.55
eon	Base	0.41	0.46	0.74	1.08	1.83
	Best	0.41	0.46	0.74	1.08	1.83
	PS	0.40	0.46	0.74	1.08	1.83
gcc	Base	0.69	1.07	1.09	1.38	1.50
	Best	0.68	1.06	1.09	1.38	1.51
	PS	0.73	1.11	1.15	1.44	1.55
	PS 50	0.68	1.06	1.10	1.37	1.48
gzip	Base	0.51	0.76	0.78	1.17	1.85
	Best	0.51	0.76	0.78	1.17	1.85
	PS	0.50	0.76	0.78	1.17	1.86
lucas	Base	0.50	0.50	0.50	0.50	0.50
	Best	0.50	0.50	0.50	0.50	0.50
	PS	0.50	0.50	0.50	0.50	0.50
mcf	Base	0.27	0.27	0.27	0.28	0.28
	Best	0.28	0.28	0.28	0.28	0.28
	PS	0.28	0.28	0.28	0.28	0.28
perl	Base	0.10	0.48	0.79	1.11	1.52
	Best	0.12	0.49	0.79	1.12	1.49
	PS	0.16	0.48	0.79	1.11	1.51

Table 7: Comparison of IPC for baseline DTS (Base), direct phase substitution (PS) and best-recent phase substitution (Best) when thermal throttling is triggered at $T_{lim} = 70, 75, 80, 85$ °C (assuming $t_{off} = 200\,\mu s$). $T_{lim} = \infty$ represents a processor that is not thermally constrained. The number of phases for direct phase substitution is given in Table 3. For benchmarks *bzip2* and *gcc*, we also show direct phase substitution when SimPoint parameters are modified to obtain a larger number of phases (92 phases for *bzip2*, 50 phases for *gcc*).

corresponding to the time-average power density of benchmark *gcc* (with appropriate scaling so that initial temperature does not exceed T_{lim}).

Benchmarks *lucas* and *mcf* are relatively "cold" and are not impacted by the temperature limit. Other benchmarks, like *art* and *bzip2*, are strongly impacted by thermal throttling. Their performance increases with T_{lim}. As can be seen, direct phase substitution provides an accurate approximation of the behavior under thermal throttling, except for *bzip2* and *gcc* where the approximation is rougher. For these two benchmarks, as shown in the table, we can obtain a more accurate approximation by configuring SimPoint so as to obtain more phases.

Results in Table 7 may look surprising. For example, when going from $T_{lim} = 70$°C to $T_{lim} = 75$ °C, the performance of *art* is multiplied by more than 3. We do not claim this example to be realistic. Yet, the reason for this counterintuitive behavior is instructive and justifies that we explain it with a simplified model. Temperature (relative to ambient) is roughly proportional to power :

$$T - T_{amb} = R_{ja} \times P$$

where R_{ja} is the junction to ambient thermal resistance and P is the total power. Under thermal throttling, the total power is

$$P = \lambda P_{on} + (1 - \lambda) P_{off}$$

where $\lambda = \frac{t_{on}}{t_{on}+t_{off}}$ is the on/off duty cycle and is a measure of performance. P_{on} and P_{off} are the power consumptions when the processor is on and off respectively. Under thermal throttling, we have $T \approx T_{lim}$, hence

$$\lambda = \frac{T_{lim} - T_{amb} - R_{ja} \times P_{off}}{R_{ja} \times (P_{on} - P_{off})}$$

Our simulation parameters give a value of $R_{ja} \times P_{off}$ (which is the temperature contribution due to non-gated dynamic power and static power) close to $70 - T_{amb}$. Going from $T_{lim} =$ 70°C to $T_{lim} = 75$°C yields a large increase for λ, hence a much higher performance. This example stresses the importance of minimizing power consumption during off periods.

6 Example : Impact of heat-sink on performance

To illustrate the flexibility of DTS, we provide an example of a thermal-centric case study that involves several thermal simulations per benchmark.

As emphasized in Section 2, unless we run an application with a stable behavior for several minutes, the heat-sink temperature seldom reaches a steady state. The performance of a heat-sink, i.e., its ability to remove heat from the processor, depends on several parameters, in particular its dimensions and the speed of the air-blowing fan. A larger heat-sink is generally more efficient than a small one, and a fan rotating faster yields a higher heat transfer coefficient. On personal computers, for user comfort, the heat-sink should be dimensioned

so that, under low processor activity, the fan speed is minimum, or even null. Low processor activity does not mean that the processor performance can be low. Under interactive utilization, the CPU activity generated by the system and by the user's commands is low on average. But for user comfort, the response to commands should be shorter than what the user can perceive. For example, let us assume we use the processor for short times, say less than 1 second, during which the processor dissipates a high power. Temperature increases quickly, but the time is too short to reach a steady state. If the next utilization is several seconds later, the heat sink temperature has enough time to cool down. Ideally, to dimension the heat-sink under such scenario, we need a model of computer/user interaction. Such modeling is out of the scope of this report though, and we have chosen to illustrate DTS with a slightly different example, where the heat-sink is optimized for continuous utilization.

We consider the 8 benchmarks listed in Table 3 for which we have obtained the IPC/energy traces as described in previous sections. The situation we have simulated is that of executing one benchmark followed by a second benchmark, both benchmarks being executed to completion. The first benchmark execution serves to create an initial thermal state for the second benchmark. What we are measuring is the performance of the second benchmark when the maximum temperature is set to $T_{lim} = 85$ °C, and the chip temperature is controlled with an on/off thermal throttling mechanism. It should be noted that some of the benchmarks listed in Table 3 run only for a few seconds when not throttled, and the thermal state at the end of the first benchmark execution may not be representative of a continuous utilization of the processor. Hence before executing the first benchmark we initialized ATMI with the steady-state corresponding to the time-average power density generated by the *gcc* benchmark.

We number the 8 benchmarks from 1 to 8. Let $j \in [1, 8]$ be the first benchmark and $i \in [1, 8]$ the second benchmark. Let $t_\infty(i)$ be the total execution time of benchmark i when there is no thermal throttling (i.e., $T_{lim} = \infty$). Let $t_{85}(i, j, h)$ be the total execution time of benchmark i when executing after benchmark j, where h is the heat-sink effective heat transfer coefficient, and under $T_{lim} = 85$ °C. Coefficient h, in W/m^2K, is related to the heat-sink thermal resistance R in K/W as follows :

$$R = \frac{1}{hA}$$

where A is the heat-sink base area. The higher h, the more efficient the heat sink. In our simulations, we assumed $A = 49\,cm^2$. We define the average processor performance as follows :

$$X(h) = \frac{1}{64} \sum_{i=1}^{8} \sum_{j=1}^{8} \frac{t_\infty(i)}{t_{85}(i, j, h)}$$

To illustrate the impact of the initial thermal state, we run another experiment similar to the one mentioned previously. We use *gcc* for the initial thermal state, but instead of simulating two applications j and i in succession, we run only benchmark i. For this second experiment, the average performance is defined as follows :

$$Y(h) = \frac{1}{8} \sum_{i=1}^{8} \frac{t_\infty(i)}{t_{85}(i,h)}$$

We have $X(h) = 1$ and $Y(h) = 1$ when there is no thermal throttling, $X(h) < 1$ and $Y(h) < 1$ otherwise. Figure 8 shows a plot of the average performance $X(h)$ and $Y(h)$ when h varies. The plot shows that both X and Y reach a maximum for $h \geq 291.5\,W/m^2K$, which corresponds to a thermal resistance $R = 0.7\,K/W$. If the 8 benchmarks we used were representative of the processor utilization, and if we wanted to dimension the heat-sink so that the fan speed is null under continuous utilization, we would conclude from this experiment that a thermal resistance $R = 0.7\,K/W$ under null fan speed is sufficient.

On this example, although using only *gcc* for the initial thermal state leads to a slight overestimation of the performance loss due to thermal throttling, this does not change our conclusion. However, it was not obvious beforehand that using *gcc* for defining the initial thermal state would be sufficient here. We had to run the simulations to convince ourselves.

This experiment is mainly an example to illustrate the gain in simulation time. The experiment to obtain the $X(h)$ curve represents a total *simulated* time of 5 hours. With our simulation infrastructure, this corresponds to 2600 hours of simulation time. This includes the time required to generate the IPC/energy traces for all 8 benchmarks [1] as well as the time required for all thermal simulations. The simulation time necessary to simulate this experiment with a temperature-aware cycle-accurate simulator would have been roughly two orders of magnitude longer than with DTS.

7 Conclusion

Simulating thermal throttling in a cycle-accurate simulator may give very different performance numbers depending on how the heat-sink temperature is initialized. We have argued in this report that there is no satisfactory way to initialize the heat-sink temperature. If we want to simulate thermal throttling, we must consider long simulations lasting tens of seconds or even several minutes. But cycle-accurate simulators are slow and make this kind of study difficult. We have proposed in this report a partial solution to this problem, decoupled thermal simulation (DTS). The basic idea, very simple, is to generate an IPC/energy trace with a cycle-accurate simulator and simulate thermal throttling in a separate thermal simulator taking as input the IPC/energy traces. The thermal simulator works at a larger time granularity than the cycle-accurate simulator and is therefore much faster. DTS is able to shorten the simulation time when a study requires to reuse the same IPC/energy trace several times. We have shown that a phase analysis tool like SimPoint can be used to decrease the trace generation time and make compact traces.

DTS, as described in this report, is not suitable for all thermal studies. DTS can be used to explore parameters that are not modeled in the cycle-accurate simulator, or parameters that can be modeled approximately in the thermal simulator. DTS as we have implemented

[1] We did not use phase substitution for this experiment.

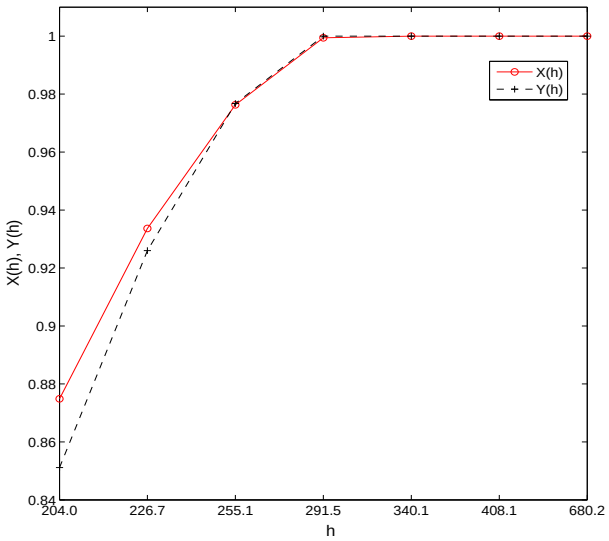

Figure 8: Average performance $X(h)$ and $Y(h)$ as a function of heat-sink effective heat transfer coefficient h.

it is only for single-thread execution and for on/off throttling. We are currently searching ways to extend DTS to multicores and to other throttling methods like dynamic voltage and frequency scaling.

References

[1] ATMI. http://www.irisa.fr/caps/projects/atmi/.

[2] F. Bellosa, S. Kellner, M. Waiz, and A. Weissel. Event-driven energy accounting for dynamic thermal management. In *Workshop on Compilers and Operating Systems for Low Power*, 2003.

[3] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2001.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch : a framework for architectural-level power analysis and optimizations. In *Proceedings of the International Symposium on Computer Architecture*, 2000.

[5] D.C. Burger, T.M. Austin, and S. Bennett. Evaluating Future Microprocessors: The Simplescalar Tool Set. Technical Report CS-TR-96-1308, University of Wisconsin-Madison, 1996.

[6] P. Chaparro, J. González, and A. González. Thermal-effective clustered microarchitectures. In *Workshop on Temperature-Aware Computer Systems*, 2004.

[7] P. Chaparro, J. González, G. Magklis, Q. Cai, and A. González. Understanding the thermal implications of multicore architectures. *IEEE Transactions on Parallel and Distributed Systems*, 18(8), August 2007.

[8] P. Chaparro, G. Magklis, J. González, and A. González. Distributing the frontend for temperature reduction. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2005.

[9] J. Choi, C.-Y. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose. Thermal-aware task scheduling at the system software level. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2007.

[10] S.W. Chung and K. Skadron. Using on-chip event counters for high-resolution real-time temperature measurement. In *Proceedings of the 10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2006.

[11] A. Cohen, L. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy. On estimating optimal performance of CPU dynamic thermal management. *Computer Architecture Letters*, 2003.

[12] J. Donald and M. Martonosi. Temperature-aware design issues for SMT and CMP architectures. In *Workshop on Complexity-Effective Design*, 2004.

[13] J. Donald and M. Martonosi. Techniques for multicore thermal management : classification and new exploration. In *Proceedings of the International Symposium on Computer Architecture*, 2006.

[14] M.S. Floyd, S. Ghiasi, T.W. Keller, K. Rajamani, F.L. Rawson, J.C. Rubio, and M.S. Ware. System power management support in the IBM POWER6 microprocessor. *IBM Journal of Research and Development*, 51(6):733–746, November 2007.

[15] S. Gunther, F. Binns, D. Carmean, and J. Hall. Managing the Impact of Increasing Microprocessor Power Consumption. In *Intel Technology Journal*, 2001.

[16] H.F. Hamann, J. Lacey, A. Weger, and J. Wakil. Spatially-resolved imaging of microprocessor power (SIMP) : hotspots in microprocessors. In *Proceedings of 10th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2006.

[17] H.F. Hamann, A. Weger, J.A. Lacey, Z. Hu, P. Bose, E. Cohen, and J. Wakil. Hotspot-limited microprocessors : direct temperature and power distribution measurements. *IEEE Jorunal of Solid-State Circuits*, 42(1):56–65, January 2007.

[18] G. Hamerly, E. Perelman, J. Lau, and B.Calder. Simpoint 3.0: Faster and more flexible program analysis. *Journal of Instruction Level Parallel*, 7, September 2005.

[19] Y. Han, I. Koren, and C.M. Krishna. Temptor : a lightweight runtime temperature monitoring tool using performance counters. In *Workshop on Temperature-Aware Computer Systems*, 2006.

[20] Y. Han, I. Koren, and C.A. Moritz. Temperature aware floorplanning. In *Workshop on Temperature-Aware Computer Systems*, 2005.

[21] T. Heath, A.P. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. Mercury and Freon : temperature emulation and management for server systems. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.

[22] S. Heo, K. Barr, and K. Asanović. Reducing power density through activity migration. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2003.

[23] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the International Symposium on Microarchitecture*, 2000.

[24] J.C. Ku, S. Ozdemir, G. Memik, and Y. Ismail. Thermal management of on-chip caches through power density minimization. In *Proceedings of the International Symposium on Microarchitecture*, 2005.

[25] A. Kumar, L. Shang, L.-S. Peh, and N.K. Jha. HybDTM : a coordinated hardware-software approach for dynamic thermal management. In *Proceedings of the Design Automation Conference*, 2006.

[26] E. Kursun, C.-Y. Cher, A. Buyuktosunoglu, and P. Bose. Investigating the effects of task scheduling on thermal behavior. In *Workshop on Temperature-Aware Computer Systems*, 2006.

[27] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *Workshop on Workload Characterization*, 2000.

[28] K.-J. Lee and K. Skadron. Using performance counters for runtime temperature sensing in high-performance processors. In *Workshop on High-Performance Power-Aware Computing*, 2005.

[29] W. Liao, L. He, and K.M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 24(7):1042–1053, July 2005.

[30] C.H. Lim, W.R. Daasch, and G. Cai. A thermal-aware superscalar microprocessor. In *Proceedings of the International Symposium on Quality Electronic Design*, 2002.

[31] Z. Lu, J. Lach, M.R. Stan, and K. Skadron. Improved thermal management with reliability banking. *IEEE Micro*, November 2005.

[32] A. Merkel, F. Bellosa, and A. Weissel. Event-driven thermal management in SMP systems. In *Workshop on Temperature-Aware Computer Systems*, 2005.

[33] P. Michaud and Y. Sazeides. ATMI : analytical model of temperature in microprocessors. In *Workshop on Modeling, Benchmarking and Simulations*, 2007.

[34] P. Michaud and Y. Sazeides. A study of thread migration in temperature-constrained multi-cores. *ACM Transactions on Architecture and Code Optimization*, 4(2), June 2007.

[35] M. Monchiero, R. Canal, and A. González. Design space exploration for multicore architectures : a power/performance/thermal view. In *Proceedings of the International Conference on Supercomputing*, 2006.

[36] M. Mutyam, F. Li, V. Narayanan, M. Kandemir, and M.J. Irwin. Compiler-directed thermal management for VLIW functional units. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tool Support for Embedded Systems*, 2006.

[37] S.H.K. Narayanan, G. Chen, M. Kandemir, and Y. Xie. Temperature-sensitive loop parallelization for chip multiprocessors. In *Proceedings of the International Conference on Computer Design*, 2005.

[38] M.D. Powell, M. Gomaa, and T.N. Vijaykumar. Heat-and-run : leveraging SMT and CMP to manage power density through the operating system. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2004.

[39] M.D. Powell, E. Schuchman, and T.N. Vijaykumar. Balancing resource utilization to mitigate power density in processor pipelines. In *Proceedings of the International Symposium on Microarchitecture*, 2005.

[40] N. Rinaldi. On the modeling of the transient thermal behavior of semiconductor devices. *IEEE Transactions on Electron Devices*, 48(12):2796–2802, December 2001.

[41] E. Rohou and M.D. Smith. Dynamically managing processor temperature and power. In *Workshop on Feedback-Directed Optimization*, 1999.

[42] K. Sankaranarayanan, S. Velusamy, M. Stan, and K. Skadron. A case for thermal-aware floorplanning at the microarchitecture level. *Journal of Instruction-Level Parallelism*, 7, October 2005.

[43] B.C Schafer, Y. Lee, and T. Kim. Temperature-aware compilation for VLIW processors. In *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007.

[44] L. Shang, L.-S. Peh, A. Kumar, and N.K. Jha. Thermal modeling, characterization and management of on-chip networks. In *Proceedings of the International Symposium on Microarchitecture*, 2004.

[45] A. Shayesteh, E. Kursun, T. Sherwood, S. Sair, and G. Reinman. Reducing the latency and area cost of core swapping through shared helper engines. In *Proceedings of the International Conference on Computer Design*, 2005.

[46] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[47] K. Skadron, T. Abdelzaher, and M.R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2002.

[48] K. Skadron, M.R. Stan, W. Huang, and S. Velusamy. Temperature-aware microarchitecture. In *Proceedings of the International Symposium on Computer Architecture*, 2003.

[49] J. Srinivasan and S.V. Adve. Predictive dynamic thermal management for multimedia applications. In *Proceedings of the International Conference on Supercomputing*, 2003.

[50] J. Srinivasan and S.V. Adve. The importance of heat-sink modeling for DTM and a correction to "Predictive DTM for Multimedia Applications". In *Workshop on Duplicating, Deconstructing and Debunking*, 2005.

[51] W. Wu, L. Jin, J. Yang, P. Liu, and S.X.-D. Tan. Efficient power modeling and software thermal sensing for runtime temperature monitoring. *ACM Transactions on Design Automation of Electronic Systems*, 12(3), August 2007.